November 1994

OS-9 International

Issue 3/94

EFFO OS9

European Forum For OS-9

8606 Greifensee, Switzerland
Fax +41 1 940 38 90
email os9int@effo.ch

sFr. 10.00
ISSN: 1019-6714

startup

Usually, you are unenlightened but blessed. At least, until a good friend sends you a Dhrystone program to measure CPU performance. The first mistake is to run this program on your MC68040 computer. The second even more relevant mistake is to tell your friend that your machine showed 12 kDhrystone, which is pretty much, isn't it? If your friend responds that this processor should give a value close to the final answer of 42 you really feel unenlightened but definitely unhappy.

The first and easiest solution is to buy the same machine as your friend owns. Don't forget to take the Dhrystone program along with you when you go out for shopping.

The second comparably expensive solution is to see your analyst. He or she will probably track down the problem to a really disastrous constellation at your early baby stage. But concerning today's excellent analysts there is a good chance that you will learn to live with your problem.

You are definitely in trouble, if neither of the above solutions is acceptable for some reason, since there is a good chance that the problem is related to on-chip caches. Unfortunately, management of the MC68040 caches is one of the least understood secrets.

The next step is to find the software engineer who ported OS-9 to your hardware. If the company is small, you are out of business, because the one and only engineer missed the point, obviously. If the company is large, there is absolutely no chance to catch just that one and only engineer who understands caching. From a medium-sized company you will get the answer, that the one and only engineer who understands caching just abandoned his job in order to join the large company. Hence, you better forget this solution.

Please don't even think about reverse engineering of drivers and file managers. It will take several weeks, if you start from scratch and, even if you succeed, you will not only feel unhappy but criminal too (anybody out there who read the license agreement lately?).

There is, at least, one solution that really works: ask for the source of the Dhrystone program and multiply a certain variable at a strategic place with the arbitrary factor of 3.2.

Now, if you ask how to proceed – I don't know. Maybe, you should consider using a completely different operating system. But even in this case, you should read the related article in this issue first.

Werner Stehling

SYSTEM-PAK I/MGR

Die Grafik-Oberfläche für OS-9/68xxx, LynxOS und VxWorks:

Maschinensteuerung
Bildverarbeitung
Meßwertdarstellung
Softwareentwicklung

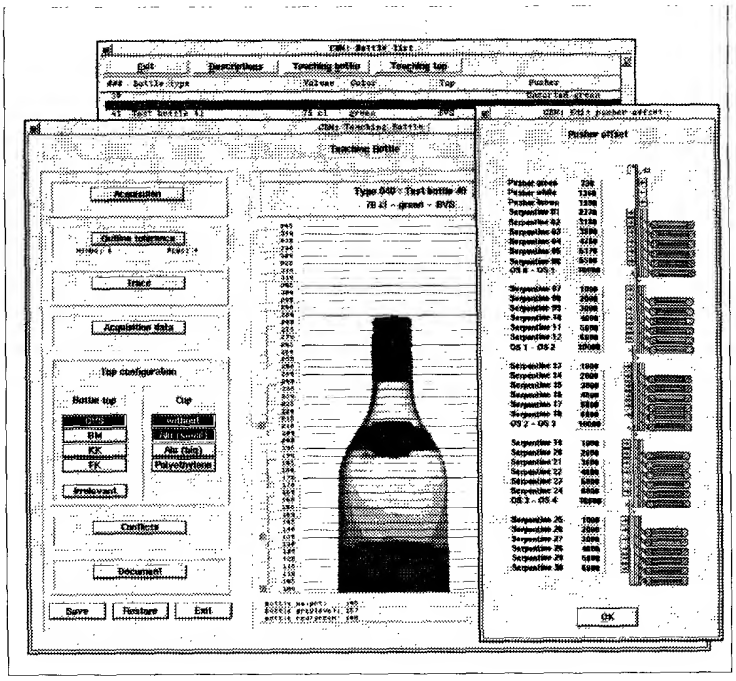

Abbildung mit freundlicher Genehmigung der CHESTON AG Automation AG, CH-5706 Boniswil

Dialog aus der "Teaching" Phase einer Flaschensortieranlage. Die Bedienoberfläche wurde mit dem MGR und der MGR/ALib Application Library realisiert.

reccoware systems

Neue Anschrift: *reccoware systems*, Wolfgang Ocker, Rapperzell, Föhrenstraße 8, D-86576 Schiltberg, Tel. (0 82 59) 10 48, Fax 10 49, Email: reccoware@recco.de

OS-9 Cross Development on the Macintosh

Lukas Zeller

Introduction

The Macintosh Programmer's Workshop (*MPW*) smartly combines the advantages of both a command line based programming environment and a windowed graphical user interface. Essentially, *MPW* consists of a multi-window editor and command line shell (the *MPW* shell) and a set of so-called *MPW* tools. These tools are invoked in a similar way as OS-9 or UNIX programs with command line options and arguments to control their behaviour. Equivalents are provided in the standard *MPW* tool set for most OS-9 utilities such as *dir* (*files*), *grep* (*search*), *diff* (*compare*) and much more. This makes *MPW* a very good platform for porting traditional command line based development tools to the Macintosh. In fact, porting a tool to *MPW* is relatively straightforward, if the C source code is available.

Why not use this environment for OS-9 cross development? This would allow using both the comfort of the Mac for development and the real-time power of OS-9 in the target system. In addition, many features of *MPW* would facilitate and speed up software development. For instance, the *Projector* provides project management and version control, the *MPW make* utility is superior to the OS-9 *make* and the editor's selection is automatically positioned on the source line that contains a syntax error.

Unfortunately, there are no OS-9 cross compilers for the *MPW* platform available so far. This situation led to the idea for *OS9exec*, since the Mac and OS-9 systems have one important thing in common: the 680xx processor. So why shouldn't it be possible to find a way to make unmodified OS-9 code run under *MPW*? Obviously, this is possible with a version of the OS-9 operating system running on the Mac, which is already commercially available. But the *OS9exec* idea is definitely different. It was intended to embed OS-9 tools in a software frame so that they can be used like standard *MPW* tools. This implies emulation of certain OS-9 services, but is strictly limited to the requirements of development tools, i.e. mainly I/O services. Multitasking and real-time features are not supported.

This article describes important aspects of *OS9exec*'s implementation and points out some of the problems encountered and solutions found during the realization of this project.

Emulating an OS-9 Runtime Environment

The following points must be considered in order to provide a run-time environment for a single OS-9 process under *MPW*:

- The program's executable module must be loaded into memory.
- The static storage requirements for the program must be calculated from the OS-9 module's header fields (*M$Data*, *M$Stack*) and allocated.
- OS-9 provides automatic initialisation of the static storage to fulfil the needs of initialised global variables in languages such as C. This task is normally done by the *F$Fork* system call and not in the program itself. *OS9exec* must, therefore, emulate this behaviour.
- Next, the command line arguments and environment variables must be passed to the OS-9 program. Unfortunately, the method used by OS-9 is only vaguely documented and has a rather bizarre structure that is difficult to implement.
- Before the OS-9 program can be launched, the initial values of all processor registers must be calculated. This includes the program counter that must point to the program's execution entry, the stack pointer, the static storage pointer and others as described in the *F$Fork* documentation. These values are stored in a C structure called *os9regs* that can be passed to the context switching routine.
- Now a context switching routine implemented in assembly language called *os9_go* has to be invoked. The main task of this routine is to save the Mac context's registers and to load the passed *os9regs* structure into the actual processor registers. In addition, the routine must take measure to return control to the caller (the *OS9exec*'s C part) whenever the OS-9 program invokes an OS-9 system call. This includes updating the register values in the *os9regs* structure and returning the OS-9 system call number.
- The system call whose number was returned by the context switching routine must now be emulated. For most system calls this consists of the following steps: the emulating code must inspect the register values in the *os9regs* structure, call appropriate Mac OS routines (e.g. file I/O) and store results back into the *os9regs* structure. Then, the *os9_go* function is called once more to continue the execution of the OS-9 program. A similar mechanism is used to cope with calls to trap handlers like *cio* and *math*.
- A special case is the *F$Exit* function that terminates the emulation and returns control to the *MPW* shell.

The following pseudo-code gives an overview about the C part of *OS9exec*:

```
os9exec()
{
  /* prepare OS-9 environment */
  load the program's executable module
  prepare for launch {
    allocate static storage
    initialise variables and data references
    prepare command line arguments and environment
    set up processor register structure os9regs
  }
  /* main loop */
  do {
    trapcode = os9_go(&os9regs)
    if (trapcode = 0) {
      emulate OS-9 system call
      modify os9regs as a result of the system call
    }
    else if (trapcode = 1..15) {
      if (appropriate trap handler is installed) {
        prepare os9regs such that next os9_go() call...
        ...continues execution in the trap handler
      }
    }
    else if (trapcode = errorexception) {
      prepare os9regs such that next os9_go() call...
      ...continues execution in the exception handler
    }
  } while (F$Exit not occurred)
  /* clean up */
  close paths, deallocate memory etc.
  return (exit status of OS-9 program)
} /* end of os9exec */
```

Context Switch

The basic problem of the *os9_go* routine is that Mac/*MPW* and OS-9 run-time environments have almost nothing in common except the 680xx processor. This requires the implementation of a context switching mechanism that allows the different methods of memory allocation to coexist.

OS-9 expects to have the program stack inside its static storage area, which is a completely different location as compared to the Macintosh stack. In contrast to OS-9, both Mac programs and system software normally run in system state and, thus, share the same stack (pointed to by the interrupt stack pointer, *ISP*) to some extent.

As a consequence, the context switching code must be able to switch all processor registers including the *ISP*. Most of the Mac's registers can be saved on the Mac stack but, after changing the *ISP*, the memory address of the Mac stack and, hence, the saved registers would get lost. A global variable is, therefore, required to store the Mac *ISP* before switching to OS-9 context.

As of today there is still an application specific area reserved, named *ApplScratch*, that can be used to store the *ISP*. It is located in the so-called low memory globals at the absolute address $0A78. Unfortunately, Apple recommends not to use low memory globals any longer so that a more sophisticated solution must be found in the future. Maybe a reader of this article will contribute a solution...

There is a second issue related to the relocation of the *ISP*: what happens, if an interrupt occurs from a Mac hardware device? First, I thought this should not be a problem, as an interrupt routine cannot depend on any register context anyway and would not even notice that the context has changed to OS-9. Unfortunately, this assumption proved wrong, as the position of the *ISP* is checked in some interrupt routines. If such occurs while executing OS-9 code, the Mac OS thinks that "stack has moved into application heap" (error #28, *dsStknHeap*) and probably crashes the Mac. I tried many ways to solve this problem, e.g. to intercept interrupts with special code that restored a Mac-like stack before entering the interrupt routine, or to use the user stack pointer (*USP*) for the OS-9 task to avoid changing the *ISP* at all. All attempts were not completely wrong, but none of them was stable enough to be usable. Hence, the only feasible but not elegant solution was to mask all interrupts as long as the program runs in OS-9 context. This may sound ugly and in fact completely blocks all background processing such as mouse pointer movement and network response. In practice, however, you hardly notice it. Fortunately, the average time between OS-9 calls which cause a context switch back to the Mac (see below) is so short that the effect for the user is similar as during floppy operations.

The above shows how the OS-9 context is entered from *MPW*. From this point, the program is running in OS-9 context until it reaches an OS-9 system call in form of a *TRAP #0* instruction. The *os9_go* routine must then return control to the C part of *OS9exec*.

This is implemented by replacing the *TRAP #0* entry in the exception table by a pointer to a special exception handler. This handler saves the OS-9 context back into the *os9regs* structure, restores the original *MPW* stack (using the global *ApplScratch* variable) and returns control to *OS9exec* providing the OS-9 system call number.

A similar mechanism is used to exit *os9_go* for *TRAP #1..#15* as well as for other exceptions like division by zero. To inform the calling program why *os9_go* has exited, its return value consists of the trap number and a function code. With this information, the C part of *OS9exec* can take the proper measures such as emulating a system call or dispatching the program flow to a trap handler (e.g. *cio*, *math* or *csl*).

Maintaining an *MPW*-like Look and Feel

As described in the introduction, the goal of the project is not to bring an OS-9-like user interface to *MPW* but to extend the *MPW* tool set with OS-9 specific compilers, assemblers, linkers etc. The best solution for this problem would be to recompile these tools for *MPW*. Unfortunately, this is not possible as the source code is unavailable. *OS9exec* fills this gap by embedding OS-9 code into *MPW* so that it finally behaves like other *MPW* tools.

This concept implies that every *OS9exec* based tool must include all OS-9 modules required to execute (e.g. *cio*, *math*, *csl*). To achieve this, OS-9 modules are stored within the *MPW* tool's resource fork in a resource called *OS9C*. Item number 0 must contain the main OS-9 module. If more modules are required, they must also be included as *OS9C* resources, with any ID except 0. It is, however, very important that the name of each *OS9C*-resource is the same as the name of the OS-9 module. *OS9exec* references modules by their *OS9C* resource names only, the real module name contained within the OS-9 module is not relevant. *MPW*'s *Rez* resource compiler easily allows the inclusion of "foreign" code into a named resource through its *read* statement.

Another issue of embedding an OS-9 program like a compiler, an assemblers or a linker into the *MPW* environment is the way error messages are displayed. Real *MPW* tools output error messages in form of *MPW* script commands (File and Line). These may be selected and executed to have the *MPW* shell open the appropriate file and select the line in error. To bring this feature to an *OS9exec* based tool, an external routine *writeline* is called whenever the program writes to *standard output* or *standard error output*. This routine must be implemented in the tool-specific source file. It should scan the program's output lines for error messages and add an *MPW*-compatible message to the output stream. The following example code shows how this is implemented for the *cc1* compiler component that produces the C-source related error messages:

```
void writeline(char *linebuf, FILE *stream)
{
   char *ptr;
   char *thisline;
   char *thisfile;

   /* - output original line first,
    *     prefixed with MPW's comment character: #
    */
   if (*linebuf!=0) {
      fputs("# ",stream);
      fputs(linebuf,stream);
   }
   putc('\n',stream);
   fflush(stream);
   /* analyse line */
   if (strstr(linebuf,"time in")==NULL) {
      if (ptr=strstr(linebuf,": ")) {     /* extract filename */
         while (*(--ptr)!=':');
         *(ptr++)='\0';                   /* set end of filename */
```

```
            thisfile=linebuf;                /* pointer to filename */
            sscanf(ptr,"%d:",&thisline);     /* extract line number */
            /* - output appropriate MPW File and Line commands */
            fputs("#----------\n",stream);
            fprintf(stream,"    File \"%s\"; Line %d\n",
                    thisfile,thisline);
            fputs("#----------\n",stream);
        }
    }
} /* writeline */
```

Finally, the exit codes of OS-9 and *MPW* have not the same meaning. Some compiler's exit values cause an *MPW* script to abort prematurely, even if the compiler has detected warnings only, not errors. To avoid this, the exit code can be filtered in the tool-specific source file.

Feature Overview

Capabilities

OS9exec in its current version

- is a framework that can be used to enhance an unmodified OS-9 compiler, assembler, linker etc. to act as an *MPW* tool. It can then be used like any other tool within the *MPW* or *ToolServer* environments.
- works with Microware's K&R C compiler as well as with *Ultra-C*, including *r68*, *l68* and *rdump*. It also works with Stephan Paschedag's port of *gcc/gpp*, with Lloyd I/O's *CRASMB* cross assembler and IAR's *icc* cross compiler system.
- supports trap handlers like *cio* or *math* as well as subroutine/data modules (like *CRASMB*s CPU modules).
- supports output filters that may convert the error messages of a compiler into *MPW* standard format (see above).
- automatically converts OS-9 paths into Mac paths, can resolve aliases, substitutes *MPW* shell variables in path lists and allows definition of a "/dd" folder through the *OS9DISK MPW* variable. It supports the OS-9 environment: all exported *MPW* shell variables whose names start with a '@' character are available in the OS-9 environment (not including the '@').
- works in the 68020 emulation of the PowerMac.
- is available as an EFFO PD. Please note, however, that it is not completely public domain: you may not use it commercially (i.e. sell it along with your OS-9 compilers to make them run on the Mac) without the author's written consent.

Limitations

OS9exec in its current version

- seems to conflict with Connectix' *RAM Doubler* although all storage used by OS-9 is locked in memory to prevent page misses in OS-9 code. Hints on how to solve this problem are welcome!
- disables all interrupts while executing OS-9 code. This stops all background processing. However, interrupts are enabled whenever an OS-9 system call is executed. As this happens quite often, the system response delay is not much worse than while formatting a floppy disk.
- emulates a singletasking OS-9 environment. None of the multitasking-related services are supported. A special exception is *F$Fork*: this is used by C executives like *cc* or *gcc* to launch their components. *OS9exec* provides a simple solution: whenever *F$Fork* is invoked, it just outputs an appropriate command line to *stdout*. This output line can then be gathered in a file and executed as a script. No doubt, it is a funny solution but it works. A complete implementation is shown in the *cc*, *gcc* and *ucc* scripts, for instance.
- supports a small subset of OS-9 system calls only, namely those being required by compiler-like programs. If a non-implemented system call is invoked, *OS9exec* reports this to *stdout* and returns *E_UnkSvc* to the caller. If a program does not work this way, because it requires this call, the addition of an appropriate emulation to *OS9exec.c* must be considered. This was the approach I used to determine the minimal set of required OS-9 calls.
- is not written in a nice programming style... There are a lot of CPU dependencies such as relying on Motorola-type byte ordering, there are type casts, and everything is in a single file etc. As mentioned in the introduction, *OS9exec* must be taken as the result of an experiment. I would be glad if readers were tempted to improve *OS9exec* with own ideas and experience.

How To Create an *OS9exec* Based Tool

You need

- *MPW* V3.3 or newer plus the *MPW* C compiler on a Macintosh running System 7 or newer. *OS9exec* might well work with older System or *MPW* versions, but it was not tested. *MPW* is available from the Apple Program Developer's Association (APDA).
- a valid developer's license of OS-9 and an appropriate computer system that allows to transfer files between OS-9 and Macintosh (e.g. pcf, network link or RS-232).

- EFFO PD #123 with the *OS9exec* software. Note that this disk is a Macintosh formatted floppy disk.

Steps to take (details see documentation of PD #123)

- Transfer recent versions of the original OS-9 header files *module.h* and *errno.h* to the Mac. Also transfer all OS-9 modules required to run the program. The latter, normally, consist of the program module plus trap handlers (*cio, math, csl* as required). As an example, we will use the OS-9 V2.x *date* utility that requires *cio.*
- Place the transferred files into the appropriate Mac folders as described in the documentation.
- PD #123 already provides dedicated source files for a number of OS-9 programs including the *date* utility in form of the *os9date.c* file. In these cases, the *OS9exec* based tool can be built directly using *MPW*'s *BuildProgram* script (or *Command-B* menu command). Note that the produced *MPW* tool's name will be *os9date* and not *date* to prevent confusion with *MPW*'s built-in *date* command.
- Move the created *MPW* tool into the *{MPW}Tools* folder.
- Use the tool. To do this for the *date* utility, type *os9date*, hit *Enter* and enjoy the time and date information.

Lukas Zeller works as software and hardware engineer and is engaged in projects related to today's social and environmental changes. He can be reached at <lukas@effo.ch>.

The GNU C Compilers, Part 2

Stephan Paschedag

Introduction

This is the second part in a series of articles about the GNU C Compiler (GCC) and the GNU C++ Compiler (GPP). The first article has been published in the previous issue of this journal [1] and gave a more general overview on the compiler capabilities. This part describes the internal functionality of the executive and the actual compiler [2].

Normally, GCC and GPP compilers are not called directly from the *shell* input line or from the *make* utility but via a front-end executive program that is named *gcc* on most computer systems. Under OS-9, the program name *gcc* is reserved for version 1.x of GCC that still is in use on smaller systems. The names of OS-9 GCC programs for version 2.x are, therefore, suffixed with 2, e.g. *gcc2*.

The Executive

The executive *gcc2* is responsible for parsing the argument line, defining the compiling procedures and calling the various compiler modules. The command line

```
gcc2 -o t t.c
```

leads, for example, to the following five consecutive steps:

1. Pre-processor

The automatically generated command line to call the pre-processor *cccp2* contains the default definitions. These include settings for the language of the source file, the operating system, the processor type and the path name of the temporary output file. The latter is assumed to be on RAM disk (*/r0*) by default. The principal function of the pre-processor is to insert all include files and to substitute the macros in the source file. In the above example, the following pre-processor command line is produced (all in one line):

```
cccp2 -lang-c -v -undef -D__GNUC__=2 -DOSK -D_OSK -D__OSK__
      -D___OSK__ -D__mc68000__ -Dmc68000 t.c /r0/cc.000021.cpp
```

2. Compiler

In a subsequent step, the compiler *cc2* is called; its assembly language output is also written to a temporary file on the default output device. This step is the most important part of the compiling procedure.

```
cc2 /r0/cc.000021.cpp -quiet -dumpbase t.c -version -o /r0/cc.000021.a
```

3. Assembler

Next, the Microware assembler *r68* is invoked. It translates the assembly source code into binary code. This code is called relocatable object file (ROF) and may still contain unresolved references.

```
r68 -v -q /r0/cc.000021.a -o=t.r
```

4. Collector

The collector (*collect*) is only needed for projects combining C and C++ modules, but it is always called unless disabled using the *-nocol* option. It locates those functions in the ROFs that initialize and deinitialize global objects. Calls to these constructor and destructor functions are placed at the beginning and at the end of subroutines referencing a particular global object. The *r68* assembler is then invoked again to translate the collector's output.

```
collect -o /r0/cc.000021.col /dd/LIB/cstart.r t.r
      -l=/dd/LIB/gclib000.l -l=/dd/LIB/math.l -l=/dd/LIB/sys.l
r68 -qo=t_col.r /r0/cc.000021.col
```

5. Linker

In a final step, the Microware linker *l68* is called in order to produce the executable program module. Its main task is to include unresolved global symbols from the *cstart.r* file and from the libraries.

```
l68 -o=/r0/t -a /dd/LIB/cstart.r t.r t_col.r -l=/dd/LIB/gclib000.l
      -l=/dd/LIB/math.l -l=/dd/LIB/sys.l
```

In principle, the type of a file is identified by its suffix so that only the remaining compile steps are performed. The executive *gcc2* is able to distinguish between two languages. C source files are recognized by the extension '.c', C++ files by '.C', '.cc' or '.cxx'. Files having the extension '.cpp' are expected to be pre-processed C files.

In addition, *gcc2* facilitates library management. Unlike Microware's *cc*, *gcc2* has built-in rules to locate and name library files. It is not necessary to specify the full path name of a library. Instead, the directory path of the library and even its '.l' suffix may be omitted. To ease support of UNIX software, the 'lib' prefix of the library name may also be omitted. In all cases, the full path name is generated according to the order defined in the library search rule.

Here is an example: the command line

```
gcc2 -lgpp t.r -o t
```

is automatically expanded to

```
168 t.r -l=/dd/lib/libgpp.l
```

The Compiler

The main part of GCC and GPP are the compilers *cc2* for the C and *cc2plus* for the C++ language, respectively. They optimize the code as defined by command line options and translate it into assembly language. Additional but less important optimizations may be done by the *o68*, *opt68k* and *r68* passes later on.

The compilers consist of two main parts, only one of them is language specific. These two parts are contained in the same executable program, while *Ultra C*, for example, uses different programs for the various stages of the compilation. This may lead to an increase in compilation time because more temporary files need to be created and accessed. The general design of GNU compilers allows to write other compilers such as C++, Objective-C, Pascal and Ada just by rewriting the language specific part. The other part that includes optimizations and code generation does not need to be rewritten.

The compilation procedure consists of separate passes that are described in the following paragraphs.

Parsing

This pass reads the entire text of a function definition and builds a syntax tree in memory. This tree does not completely follow C syntax, because it is intended to support other languages as well. Simplifications of arithmetic expressions and constant folding are also done during this pass. The latter represents an optimization technique that deals with constants. For example, the expression `x*0` is replaced by `0` and `A & 2 ? 2 : 0` is reduced to `A&2`.

Generation of an Intermediate Language

GCC compiles the source code via an intermediate language called Register Transfer Language (RTL). This language describes every single instruction of the source code in an algebraic form. It can be inspected, if *gcc2* is started with the *-dr* option. The name of the RTL file is made by appending '.rtl' to the input file name. Here is an example of an RTL statement :

```
(insn 9 8 11 (set (reg/v:SI 28)
        (const_int 5)) -1 (nil)
    (nil))
```

This statement sets the integer register number 28 to the constant value 5. This register number is completely independent of the processor's internal registers. All variables are stored in so-called pseudo-registers. It is only later in the process of compilation that they are assigned either to one of the processor's internal registers or to stack addresses. The RTL pass already optimizes 'if'-conditions that are comparisons, Boolean operations and conditional expressions. At this time, situations where a subroutine makes a recursive call to itself at its end (tail recursion) are also detected. In addition, decisions are made how to arrange loops and how to output switch statements. The latter can either be implemented as a jump-table or as a binary-tree of comparisons. A jump table always executes faster but its size may become too large, if only a few cases are used in a large span of case values.

Another optimization procedure deals with subroutine calls. If a program contains only a few number of branches to a short subroutine, the entire subroutine may better be inserted at the place of the branch statement. This is called 'function inlining' and avoids the required stack operations for parameter passing to the subroutine thus increasing the program's execution speed. It is, however, only justified if other global optimization procedures are enabled so that the increase in program length does not counteract the speed increase of inlining. The latter, however, can only happen if instruction cache is enabled and the inlined code no longer fits into cache.

Principally, the function to be inlined may contain all constructs supported by GCC such as gotos, loops, recursive calls to itself and even tail-recursive functions. The decision of whether a particular function is inlined is made at the end of RTL generation.

Jump Optimization

This pass eliminates unnecessary jump statements if, for example, they point to another jump statement. In addition, unreferenced labels and code segments are deleted. These steps of jump optimization are performed up to three times, immediately following RTL generation, after common subexpression elimination and before the final pass.

Register Scan

The first and the last use of every pseudo-register are determined in order to optimize their assignment to the processor's internal registers. This assignment does not need to be permanent throughout a function, but it may change from block to block depending on the usage of these registers.

Jump Threading

This pass detects a conditional jump that branches to an identical or inverse tests. They are replaced by unconditional jumps to the second test.

Common Subexpression Elimination

The basic idea of common subexpression elimination (CSE) is to replace or simplify a sequence of operations by equivalent ones that require less computing time. For example, the code segment

```
int a;
abc(int i)
{
  if (i > 100)
    a = i * 27351;
  else
    a = -i * 27351;
}
```

contains twice a multiplication with 27,351. The compiler is able to rearrange the instructions so that the multiplication is only needed once.

Instruction Optimization

The above example may also serve to exemplify instruction optimization. GCC optionally replaces the time-expensive multiplication instruction by a faster, but longer sequence of add, subtract and shift instructions. It may also take advantage of the addressing capabilities of the target processor. The MC68020, for example, is able to directly address 8-bit, 16-bit, 32-bit and 64-bit elements of arrays by allowing for a scaling factor. The instruction

```
lea (a0,d0*2),a0
```

calculates an effective address which is the sum of the base-register *a0* and the index register *d0*, scaled (i.e. multiplied) by 2. This instruction could be used to address an element in a one-dimensional array of short integers. Since these instructions represent highly efficient combi-

nations of several arithmetic operations, GCC uses them also for other purposes. If, for example, a C source code contains the expression

```
a0 = a0 + d0 * 2;
```

it can similarly be compiled to the above single-line instruction. GCC is even able to combine several of them as can be seen in the following assembly output of the code segment from the previous paragraph:

```
abc:
    move.l d2,-(a7)                 ; save register d2 to stack
    move.l d0,d1                    ; copy value of the first function argument i
    moveq.l #100,d2
    cmp.l d1,d2
    blt L4.                         ; branch if (i > 100)
    neg.l d1
L4.
    move.l d1,a0                    ; copy i to a0
    lea (a0,d1.l*2),a0              ; a0 = i + 2*i = 3*i
    move.l a0,d0                    ; copy 3*i to d0
    asl.l #5,d0                     ; d0 = 3*i * 32 = 96*i
    sub.l d1,d0                     ; d0 = 96*i - i = 95*i
    asl.l #5,d0                     ; d0 = 95*i * 32 = 3040*i
    sub.l d1,d0                     ; d0 = 3040*i - i = 3039*i
    move.l d0,a0
    lea (a0,d0.l*8),a0              ; a0 = 3039*i + 3039*i * 8= 27351*i
    move.l a0,a(a6)                 ; store result to global variable a
    move.l (a7)+,d2                 ; restore register d2 from stack
    rts
```

Another example shows that the scaling factor in addressing modes may also be used to multiply a variable and push the result onto the stack in a single instruction. The C language function call

```
abc(1, 2, i*5);
```

is compiled to

```
    move.l i(a6),a0                 ; get the global variable i
    pea (a0,a0.l*4)                 ; push the result of i + i*4 = i*5
    moveq.l #2,d1                   ; prepare passing of second argument
    moveq.l #1,d0                   ; prepare passing of first argument
    bsr abc
```

Loop Optimization

This pass moves constant expressions out of loops. A benchmark program may contain the following sequence to determine the time a particular processor needs for 32-bit multiplication:

```
abc(short i)
{
  int f,g;

  while (i--) {
    f = 1234723*g;
  }

  def(f);
}
```

GCC will fool this benchmark program, since it produces the assembly sequence

```
abc:
    move.l d2,-(a7)                 ; save register d2 to stack
    subq.w #1,d0                    ; needed for dbra instruction
    cmpi.w #-1,d0
    beq L3.                         ; skip loop if i is zero
    move.l d2,d1                    ; g used uninitialized in this case
    muls.l #1234723,d1              ; f = 1234723*g
L4.
    dbra d0,L4.                     ; empty loop
L3.
    move.l d1,d0                    ; prepare passing of argument f
    bsr def
    move.l (a7)+,d2                 ; restore register d2 from stack
    rts
```

that moves the multiplication out of the loop so that the execution time of an empty loop is determined.

In addition, this pass optionally includes two other optimization strategies: strength reduction and loop unrolling.

Strength reduction changes a time consuming arithmetic operation into a less consuming one. For example, multiplication may be changed to a shift or an add operation, or division by a constant is changed to multiplication with the reciprocal of the constant.

The loop unrolling technique can be used, if the loop counter is a constant and has a relatively small value. In this case, the loop construct is removed and the loop's content is repeated as often as required by the loop count. Obviously, loop unrolling may enlarge code size considerably. In principle, the effect is very similar to function inlining and the use of pre-processor macros.

Data Flow Analysis

This pass divides the program into functional (so-called basic) blocks. Among others, computations whose results are never used are deleted. If, in the above example, the call to the 'def' function were omitted, the compiler's assembly output would have the following form:

```
abc:
    subq.w #1,d0                    ; needed for dbra instruction
    cmpi.w #-1,d0
    beq L3.                         ; skip loop if i is zero
L4.
    dbra d0,L4.                     ; empty loop
L3.
    rts
```

This pass also checks whether a memory referencing instruction is preceded or followed by an instruction that increments or decrements the memory pointer. In such cases, the two instructions may efficiently be combined into a single instruction using auto increment or auto decrement addressing.

Instruction Combination

This pass attempts to combine groups of two or three instructions that are related by data flow into single instructions. For example, the assembly sequence

```
move.l 12(a0),d0
move.l d0,36(a1)
```

can be combined into

```
move.l 12(a0),36(a1)
```

More specifically, this optimization procedure combines the RTL expressions for the instructions by substitution, simplifies the result using algebra, and then attempts to match the result against the machine description.

Register Class Preferencing

GCC scans the RTL code in order to identify the register class that is best suitable for each pseudo-register. The 68k processor family offers up to three different register classes: data registers, address registers and floating-point registers.

Register Allocation

In a first step, pseudo-registers that are used only within the same basic block are assigned to the processor's internal registers. This ensures most efficient use of the processor's hardware resources. Thereafter, the remaining pseudo-registers being used in more than one basic block are assigned to hardware registers.

Register Reloading

Pseudo-registers that could not be assigned to a hardware register are assigned to stack slots. Subsequently, instructions are identified that are invalid because a value has failed to end up in a register, or has ended up in a register of the wrong kind. These instructions are fixed by reloading the offensive values temporarily into registers. Encapsulating instructions are generated additionally to do the copying from register to memory and vice versa.

The reload pass also eliminates the frame pointer, if this is enabled by using the *-fomit-frame-pointer* option and inserts instructions to save and restore those registers that may be clobbered. Jump optimization is repeated, this time cross jump instructions are eliminated.

In addition, redundant move instructions such as *move d0,d0* are deleted.

Final Pass

One of the tasks of the final pass is to identify and to eliminate those test and compare instructions which produce a result that is never evaluated.

Furthermore, the so called peephole optimization is performed. This machine dependent technique is able to eliminate redundant instructions that were not obvious in the data flow analysis. For instance, previous more general optimization steps may need to be tuned or even be discarded in view of the properties of the specific processor. The assembly code segment

```
bsr _foobar                          ; subroutine call
addq.l #4,a7                         ; stack correction
move.l d1,-(a7)
move.l d0,-(a7)
fmove.d (a7)+,fp0
```

may further be simplified into

```
bsr _foobar                          ; subroutine call
move.l d1,(a7)
move.l d0,-(a7)
fmove.d (a7)+,fp0
```

This optimization may not have been detectable at an earlier stage, since the subroutine call and the stack correction are part of another basic block than the subsequent lines. In addition, it could have been possible that the auto incrementing addressing mode was only introduced by a previous optimization step.

The last but one step of the final pass includes the insertion of function entry and exit sequences. They are generated directly in assembly code; they never exist in the Register Transfer Language.

Finally, the output in assembly language is written to a temporary disk file for further processing by the assembler.

Debugging Information Output

If the *-gg* or *-gsrcdbg* option is specified, another output file is generated that has the file name extension '.dbg'. This file contains information for source level debugging. Currently, *gcc2* supports only Microware's *srcdbg*.

References

[1] Paschedag, Stephan, *The GNU C Compilers*, OS-9 International 2/94, page 13.

[2] Stallman, Richard M., *Using and Porting GNU CC*, Free Software Foundation, 1993, Cambridge MA.

The third and final article in this series will describe in detail how the GCC port is made and what extensions have been added that are specific to OS-9.

Stephan Paschedag works as a hardware and software engineer for a Swiss company. He can be reached by email at <stp@effo.ch>.

On-chip Caches on Motorola Processors

Carsten Emde

Introduction

The increase in computing power from Motorola's 68030 to the 68040 processor was very impressive – it was probably the greatest increase ever achieved from one processor generation to the next one. It must, however, be admitted that most of this increase in performance is due to the on-chip caches. The following table presents a comparison of processing speed values that are obtained with various Motorola processors with and without cache.

Processor		**68020**	**68030**	**68040**	**68040**	**68060**
Clock frequency		20 MHz	25 MHz	25 MHz	33 MHz	25 MHz
Dhrystone	Cache off	5,000	6,000	8,000	10,000	12,000
	Cache on	6,000	8,000	39,000	50,000	95,000

The principle of the cache mechanism is based on the fact that it takes much longer to access external memory than to access on-chip memory and registers. If cache is enabled in read mode, the content of a memory location that has recently been read is stored in cache memory; a subsequent read operation from the same memory location may then execute faster, since the data can be taken from cache memory and not from the slower external memory. If cache is enabled in write-behind mode, data are only written to external memory if there is no more cache memory available or the contents of the cache memory are explicitly flushed.

Introduction of on-chip caches has, unfortunately, a number of implications on programming strategies, especially if drivers, DMA devices or even multi-processing, dual-ported RAM, etc. are concerned. Software that makes use of one of these techniques needs to be adapted so that it correctly handles enabled on-chip caches. In fact, there are still a number of carelessly implemented OS-9 systems on the market that are built around the 68040 processor but run with the performance of a 68030-based system. This happens because the system either is not cache compatible or the system integrator did not know how to appropriately enable the caches.

The aim of this article is, therefore, to explain how the caches are managed both on the processor and on the OS-9 level. In addition, source codes of a utility program, a driver and a system-state trap handler are presented. These are used to query and to manipulate the cache settings of 68020, 68030 and 68040 processors. The 68060 processor is not yet widely available so that it is not further covered in this article.

A driver and a system-state trap handler are required, since the cache control and other cache-related registers are read using the *movec* instruction that can only be executed when the processor runs in supervisor mode. Two system-state modules instead of only one are necessary, because OS-9 has a compatibility switch that allows to selectively disable the data cache during I/O. The cache setting during I/O is obtained from the driver and the cache setting in general is obtained from the system-state trap handler.

Cache Management at the Processor Level

The cache management at the processor level is realized by a special register, the cache control register *CACR*. The bit definitions of the *CACR* are different for the 68020, the 68030 and the 68040 processor.

68020, 68030

The cache control register is 32 bits long; bits 0 to 4 and bits 8 to 13 are used, the others are reserved for future use and should not be set during write access. The data cache bits are undefined in the 68020 processor, since this processor type has no data cache.

Bit	**Function**
0	Enable instruction cache
1	Freeze instruction cache
2	Clear entry in instruction cache
3	Clear instruction cache
4	Instruction burst enable
...	
8	Enable data cache
9	Freeze data cache
10	Clear entry in data cache
11	Clear data cache
12	Data burst enable
13	Write allocate

For the purpose of the program described below, i.e. to query whether the cache is currently switched on or off, only the enable bits are important. In accordance to the above-given definitions, the bit masks to test instruction and data caches of the 68030 processor are 0x00000001 and 0x00000100, respectively.

68040

The 68040 cache control register is much simpler, only two bits are defined.

Bit	Function
15	Enable instruction cache
31	Enable data cache

A hardware reset of the 68040 processor clears the cache control register thus disabling the caches but it does not affect the other cache-related data. If the contents of the caches are valid, prior to enabling the caches they must, therefore, be invalidated using the *cinv* instruction. The setting of the cache control register does not affect the write cache mode, i.e. whether write-through or write-behind mode is selected. This is done using the MMU's transparent translation registers. They not only set the write cache to a given mode but they also allow to limit a particular write cache mode to a 16-MByte area within the 4-GByte address space. There are two transparent translation registers for data and instruction cache, respectively. They are called *DTT0*, *DTT1* and *ITT0*, *ITT1*.

The bit masks for querying and setting instruction and data cache in the *CACR* are 0x00008000 and 0x80000000, respectively.

Cache Management on the OS-9 Level

The problem of cache management in a multi-tasking environment such as OS-9 is that the cache setting must be consistent within a task, but it is not acceptable that the caches are invalidated or flushed at every task switch. In addition, the functions to manipulate the cache should be independent from the processor in use so that hardware-independent software can be written. Microware has found a solution to this problem: OS-9 uses an extension module that installs a system service request to manipulate the caches. This request is called *F$CCtl* and the extension module is called *syscache*. The latter is normally provided by the hardware manufacturer. To define the write cache mode of the 68040 processor, the already existing MMU extension module *ssm* was expanded. The transparent translation registers are set in the *ssm* module and not in the *syscache* module.

When a process calls the *F$CCtl* syscache function to disable a particular cache and this cache is currently enabled, it will be disabled immediately. In addition, a system global variable is incremented to store the number of times this cache has been disabled. This is called the disable depth, the global variables are called *D_DisData* and *D_DisInst* for data and instruction cache, respectively. Whenever the same or another process requests the particular cache to be re-enabled, the global disable depth counter is merely decremented. Only if it reaches zero the

cache is re-enabled. It is, therefore, mandatory that every process makes exactly the same number of requests to enable the cache as to disable it. If a process disables the cache only once more than it enables it and exits then, the cache will never be re-enabled and an expensive 68040 system may run only slightly faster than a 68030 system. In such a case of an erroneously unbalanced number of disable and enable calls, the program described below may need to be executed several times, in order to successfully enable the caches. When using it to disable the caches, a single execution is always sufficient.

In addition to the global disable depth variables, there is another global variable that is related to the caches, the *D_CachMode* variable. It is intended to reflect the current setting of the processor's cache control and is updated by the *F$CCtl* function. The advantage of this procedure is that the *D_CachMode* variable can, in contrast to the processor's cache control register, also be accessed in user mode; the disadvantage is that it is dependent on the correct implementation of the *F$CCtl* function. The below-given program, therefore, displays both the processor's cache control register and the current setting of the global *D_CachMode* variable.

Source Codes for Driver, System-State Trap Handler and Program

Header Files

Two header files are required, *xttn.d* for files in assembly language and *xttn.h* for files in C language.

xttn.d:

```
 org 0

_DTT0 do.l 1
_ITT0 do.l 1
_DTT1 do.l 1
_ITT1 do.l 1
```

xttn.h:

```
#define XTTNMAX 4

typedef struct xttnbits {
  unsigned int base   : 8;  /* logical address base */
  unsigned int mask   : 8;  /* logical address mask */
  unsigned int enable : 1;  /* transparent translation */
  unsigned int super  : 2;  /* supervisor/user mode access */
  unsigned int res12  : 3;
  unsigned int u1     : 1;  /* user page attribute */
  unsigned int u0     : 1;  /* user page attribute */
  unsigned int res7   : 1;
```

```
  unsigned int cm     : 2;  /* cache mode */
  unsigned int res4   : 2;
  unsigned int wp     : 1;  /* write protect */
  unsigned int res1   : 2;
} XTTNBITS;

char *xttnstr[] = {
  "DTT0",
  "ITT0",
  "DTT1",
  "ITT1"
  };

char *su[] = {
  "user ",
  "super",
  " any ",
  " any "
};

char *cm[] = {
  "cache/write-through",
  "  cache/copyback   ",
  "no cache/serialized",
  "     no cache      "
};

char *wp[] = {
  "read/write",
  "read only "
};
```

Driver to Inspect the Cache Control Register

This driver does not provide more than just the *Read* function. *Write*, *GetStat* and *SetStat* all return with error (unknown service code), *Init* and *Term* provide little or no functionality. The only purpose of this driver is to query the cache control register during I/O. The *Read* call returns the current status of the *CACR* register - a single byte at a time, as appropriate for an SCF driver. The static storage variable that holds the byte shift required for the next byte is set to 24 when the driver is initialized. The driver is called *sccctl*, the descriptor *cctl*. The *Read* call is normally made from a program. It is, however, also possible to inspect the cache control register just by loading driver and descriptor into memory and entering

```
$ tmode nopause
$ dump /cctl -c
```

so that the current *CACR* setting is displayed continuously on screen. As an advantage, the *CACR* setting may easily be inspected using this method; as a disadvantage, however, the four *CACR* bytes are not read simultaneously.

The following source code has the typical structure of an OS-9 driver. The entry points to an offset table that contains the addresses of the six mandatory subroutines.

```
Edition equ 1 current Edition number

Typ_Lang set (Drivr<<8)+Objct
Attr_Rev set ((ReEnt+SupStat)<<8)+0 Attributes and Revision

 psect SCCCTL,Typ_Lang,Attr_Rev,Edition,0,SCCCTLEnt

 use defsfile

 pag

***************
* Static storage requirements

 vsect
ByteShift ds.b 1
 ends

***************
* Module Header

SCCCTLEnt: dc.w Init
 dc.w Read
 dc.w Write
 dc.w GetStat
 dc.w PutStat
 dc.w TrmNat
 dc.w 0 Exception handler (0=none)


 pag
***************
* Initialize SCCCTL
Init:
 move.b #24,ByteShift(a2) start with MSB
 moveq.l #0,d1 no error
 rts

***************
* Read the current status of the cache control register
Read:
 movec cacr,d0
 move.b ByteShift(a2),d1
 lsr.l d1,d0
 subi.b #8,ByteShift(a2)
 bge.s Read20
 move.b #24,ByteShift(a2) start with MSB next time
Read20 moveq.l #0,d1 no error
 rts

***************
* The sccctl does not write
```

```
Write:
 bra.s PutStat
 rts

***************
* GetStat/PutStat not available
GetStat:
PutStat:
 move.w #E$UnkSvc,d1 Unknown service code
 ori.b #Carry,ccr return Carry set
 rts


***************
* Subroutine TrmNat
TrmNat:
 moveq.l #0,d1 no error
 rts

 ends
```

System-State Trap Handler to Inspect Cache Control and Transparent Translation Registers

The only relevant part of the code is the trap routine that provides the *get_cacr* and the *get_xttn* functions to be called from C level. The *get_cacr* function returns the current setting of the cache control register, the *get_xttn* function fills a structure with the four transparent translation registers.

```
 use "xttn.d"

 psect cctrap_a,0,0,0,0,0

get_cacr:
 movec cacr,d0
 rts

get_xttn:
 move.l d0,a0
 movec dtt0,d0
 move.l d0,_DTT0(a0)
 movec itt0,d0
 move.l d0,_ITT0(a0)
 movec dtt1,d0
 move.l d0,_DTT1(a0)
 movec itt1,d0
 move.l d0,_ITT1(a0)
 rts

 ends
```

The remaining code modules are, in principle, not different from the famous example in the */dd/C/SOURCE* directory that is available on every OS-9 development system. Therefore, they

are not presented here, but they are included on the OS-9 International code disk. The *trapdefs.a* binding file has only two function entries. We use trap number 4:

```
 nam CCTrap Trap code definitions

*
* This psect contains the offsets for the trap
* handler's dispatch table.  Each entry here MUST
* match the order of the dispatch table entries.
*
 psect ccdefs_a,0,0,0,0,0

* user trap vector number for CCTrap
CC$Trap: equ 4

CC$cacr: do.b 1
CC$xttn: do.b 1

 ends
```

The binding between C language and the trap handler is done in the module *cctraplib.a*:

```
 nam CCTrap Support Trap calls

cctrap macro
 tcall CC$Trap,\1
 endm

*
* This psect contains what the linker thinks is
* the code for the library reference.
*

 psect cctraplib_a,0,0,0,0,0

get_cacr: cctrap CC$cacr
 rts

get_xttn: cctrap CC$xttn
 rts

 ends
```

Program to Display and Manipulate the Cache Settings

The program *cpucache* provides both inspection and manipulation of the cache control register. If no arguments are specified, the CPU type in use, the cache-relevant compatibility variables of the system configuration module and the current *CACR* setting are analysed and displayed on screen. The latter is also given in hexadecimal notation. In addition, the current setting of the global variable *D_CachMode* and the disable depths are displayed.

The command line arguments *on* or *off* can be specified in order to enable or disable the on-chip caches. The options '-d' and '-i' are used to exclusively manipulate data or instruction cache, respectively. By default, both caches are affected. The '-q' option is provided to suppress output in change mode (quiet). If running on the 68040 processor, another option is available: the '-t' option enables the analysis of the four transparent translation registers. Here is the source code:

```
#include <stdio.h>
#include <errno.h>
#include <modes.h>
#include <module.h>
#include <setsys.h>
#include "xttn.h"

#define ENDATA   0x01
#define DISDATA  0x02
#define FLDATA   0x04
#define ENINST   0x10
#define DISINST  0x20
#define FLINST   0x40

#define CCTLNAME "/cctl"

#define DATAMASK030 0x00000100
#define INSTMASK030 0x00000001

#define DATAMASK040 0x80000000
#define INSTMASK040 0x00008000

#define CP2_DDIO 0x80

/*
 * c p u c a c h e
 *
 * switch CPU cache on and off
 */
main(argc, argv)
int   argc;
char *argv[];
{
  char             *action = NULL;
  char             *cachestr;
  char              line[52];
  int               i, j;
  int               act = -1, quiet = 0, tt = 0;
  int               cctlarg = ENDATA|ENINST;
  unsigned long     datamask, instmask, cpu;

  for (i = 1; i < argc; i++)
    if (argv[i][0] == '-')
      for (j = 1; j < strlen(argv[i]); j++)
        switch (argv[i][j]) {
          case '?':
            usage();
            exit(1);
            break;
```

```
            case 'b':
              cctlarg = ENDATA|ENINST;
              break;

            case 'd':
            case 'i':
              if (cctlarg != (ENDATA|ENINST))
                exit(_errmsg(1, "conflicting options.\n"));

              if (argv[i][j] == 'd')
                cctlarg = ENDATA;
              else
                cctlarg = ENINST;
              break;

            case 'q':
              quiet = 1;
              break;

            case 't':
              tt = 1;
              break;

            default:
              usage();
              exit(_errmsg(1, "unknown option '%c'.\n", argv[i][j]));
          }

      else
        if (action != NULL) {
          usage();
          exit(_errmsg(1, "only one parameter allowed.\n"));
        } else
          action = argv[i];

    if (action == NULL && cctlarg != (ENDATA|ENINST))
      exit(_errmsg(1, "don't know what to do.\n"));

    if (action != NULL) {
      if(!strcmp(action, "on"))
        act = 1;
      else if(!strcmp(action, "off"))
        act = 0;
      else {
        usage();
        exit(_errmsg(1, "argument '%s' not understood.\n", action));
      }
    }

    cpu = _getsys(D_MPUType, sizeof(cpu));
      if (cpu != 68020 && cpu != 68030 && cpu != 68040)
      exit(_errmsg(1, "CPU type (%d) not supported.\n", cpu));

    switch (cctlarg) {
      case ENINST:
        cachestr = "Instruction cache";
        break;
```

```
    case ENDATA:
      cachestr = "Data cache";
      break;
    case ENDATA|ENINST:
      cachestr = "Both caches";
      break;
  }

  if (cpu == 68020) {
    if (action != NULL && (cctlarg == ENDATA))
      exit(_errmsg(1, "sorry, the 68020 processor has no data cache\n"));
  }
  if (cpu == 68020 || cpu == 68030) {
    datamask = DATAMASK030;
    instmask = INSTMASK030;
  }
  if (cpu == 68040) {
    datamask = DATAMASK040;
    instmask = INSTMASK040;
  }

  if ((cpu == 68020 || cpu == 68030) && tt)
    exit(_errmsg(1, "sorry, this processor (%d) has no TT registers\n", cpu));

  if (act < 0 || !quiet)
    showcompat(cpu);

  if (act >= 0) {
    memset(line, '-', sizeof(line));
    line[sizeof(line) - 1] = '\0';
    if (act == 1) {
      if (!quiet) {
        showcacr(datamask, instmask, 1);
        printf("   ");
        printf(line);
        printf(" %s -> ON\n", cachestr);
      }
      makesuper();
      f_cctl(cctlarg);
    }

    if (act == 0) {
      if (!quiet) {
        showcacr(datamask, instmask, 1);
        printf("   ");
        printf(line);
        printf(" %s -> OFF\n", cachestr);
      }
      makesuper();
      f_cctl((cctlarg << 1) | FLDATA | FLINST);
    }
  }

  if (act < 0 || !quiet)
    showcacr(datamask, instmask, (act < 0) ? 1 : 0);
```

```
  if (tt)
    showxttn();
}


/*
 * s h o w c o m p a t
 */
showcompat(cpu)
int cpu;
{
  mod_config *config;

   /* Analyse the cache-relevant settings of init's compat bytes */
  if ((config = (mod_config *)
    modlink("init", mktypelang(MT_SYSTEM, ML_ANY))) == (mod_config *) -1)
    exit(_errmsg(errno, "can't link to system configuration module due to "));
  printf("This %d system %s disable data cache during I/O.\n",
    cpu, (config->_mcompat2 & CP2_DDIO) ? "does NOT" : "DOES");
  if (cpu == 68030)
    printf("It %s enable 68030 burst-fill mode.\n",
      (config->_mcompat & CP_NOBURST) ? "does NOT" : "DOES");
  printf("\n");
}


#define onoff(mask) ((cacr&mask) ? on : off)
/*
 * s h o w c a c r
 */
showcacr(datamask, instmask, header)
unsigned long datamask, instmask;
int           header;
{
  char          *cctlname = CCTLNAME;
  char          *on = "enabled ", *off = "disabled";
  int            cctl;
  unsigned long  cacr;

  /* Get cacr from system-state trap handler to reflect cache in general */
  cacr = get_cacr();
  if (header) {
    printf("Cache control register:\n");
    printf("   State       Value     Data cache  Instruction cache\n");
  }
  printf("   No I/O   0x%08X    %s       %s\n",
    cacr, onoff(datamask), onoff(instmask));

  /* Get cacr from driver to reflect cache during I/O */
  if ((cctl = open(cctlname, S_IREAD)) == -1)
    exit(_errmsg(errno, "can't open cctl device '%s' due to ", cctlname));
  if (read(cctl, &cacr, sizeof(cacr)) != sizeof(cacr))
    exit(_errmsg(errno, "can't read from cctl device '%s' due to ", cctlname));
  close(cctl);
  printf("    I/O     0x%08X    %s       %s\n",
    cacr, onoff(datamask), onoff(instmask));
```

```
  /* Get cacr from global variable to check its correct setting */
  cacr = _getsys(D_CachMode, sizeof(cacr));
  printf("   Global    0x%08X     %s        %s\n",
    cacr, onoff(datamask), onoff(instmask));

  /* Get global disable depth variables */
  printf("   Depth                %6d         %6d\n",
    _getsys(D_DisData, 4), _getsys(D_DisInst, 4));

}


/*
 * s h o w x t t n
 */
showxttn()
{
  int      i;
  XTTNBITS xttn[4];

  /* Get TT registers from system-state trap handler */
  get_xttn(xttn);

  printf("\n");
  printf("Transparent translation registers:\n");
  printf(
"  Register  Base   Mask  Enable Super U1 U0       Cache mode      Read/Write\n");
  for (i = 0; i < XTTNMAX; i++)
    printf("    %s    0x%02X   0x%02X    %1d    %s  %1d  %1d  %s  %s\n",
      xttnstr[i], xttn[i].base, xttn[i].mask, xttn[i].enable, su[xttn[i].super],
      xttn[i].u1, xttn[i].u0, cm[xttn[i].cm], wp[xttn[i].wp]);
}


/*
 * m a k e s u p e r
 */
makesuper()
{
  if (setuid(0) == -1)
    exit(_errmsg(1, "can't become super user.\n"));
}


/*
 * u s a g e
 */
usage()
{
  fputs("Syntax: cpucache [on|off]\n", stderr);
  fputs("Function: display/switch 68020/68030/68040 on-chip caches\n", stderr);
  fputs("Options:\n", stderr);
  fputs("    -b    both caches (default)\n", stderr);
  fputs("    -d    data cache\n", stderr);
```

```
  fputs("    -i    instruction cache\n", stderr);
  fputs("    -q    switch caches silently\n", stderr);
  fputs("    -t    also show 68040 TT registers\n", stderr);
}


/*
 * f _ c c t l
 */
#asm
f_cctl
 os9 F$CCtl
 rts
#endasm
```

Making Driver, Trap Handler and Program

The common *make* environment takes care of compiling and linking trap handler, driver, descriptor and program correctly. In addition, these four modules are merged together into a single executable program so that trap handler, driver and descriptor do not need to be loaded separately before the program can be executed. At the end of program execution, they are automatically removed from memory.

The makefile assumes that the source code environments for trap handler, driver, descriptor and program are located in the sub directories *TRAP*, *DRVR*, *DESC* and *SRC*, respectively.

```
TARGET = cpucache
DEBUG  =

RDIR   = RELS
ODIR   = CMDS

OBJS   = $(ODIR)/OBJS
SCCCTL = $(OBJS)/sccctl
CCTL   = $(OBJS)/cctl
CCTRAP = $(OBJS)/cctrap
PROG   = $(OBJS)/$(TARGET)

all:     $(CCTRAP) $(SCCCTL) $(CCTL) $(PROG)
  @merge $(PROG) $(SCCCTL) $(CCTL) $(CCTRAP) >-$(ODIR)/$(TARGET)
  @attr  $(ODIR)/$(TARGET) -a -e
  @echo  "Program 'cpucache' made."

-bu

$(CCTRAP): none%
  @chd TRAP; make DEBUG=$(DEBUG)

$(PROG):   none%
  @chd SRC; make DEBUG=$(DEBUG)

$(SCCCTL): none%
  @chd DRVR; make DEBUG=$(DEBUG)
```

```
$(CCTL):    none%
  @chd DESC; make DEBUG=$(DEBUG)
```

Limitation of the *cpucache* Program

The *cpucache* program is not intended as a general solution to a cache problem. It is primarily intended as an aid to analyse the cache setting of a system that is suspected to be slower than it could be. If an inappropriate cache setting can be substantiated, it is normally the hardware manufacturer who should provide help. This includes an update of the *syscache* and the *ssm* module, or a modified version of firmware that makes appropriate calls to the *F$CCtl* function.

Example of Program Output

```
$ cpucache -t
Cache control register:
   State       Value      Data Cache    Instruction Cache
   No I/O   0x80008000     enabled          enabled
    I/O     0x80008000     enabled          enabled
   Global   0x80008000     enabled          enabled
   Depth                      0                0

Transparent translation registers:
  Register  Base   Mask  Enable Super U1 U0       Cache Mode       Read/Write
    DTT0    0xFE   0x01    1    super  0  0  no cache/serialized  read/write
    ITT0    0xFE   0x01    1    super  0  0  cache/write-through  read/write
    DTT1    0x00   0xFF    1     any   0  0    cache/copyback     read/write
    ITT1    0x00   0xFF    1     any   0  0  cache/write-through  read/write
```

The above program output was obtained on a 68040 CPU. Write-through instruction cache is enabled in the entire 4-GByte address space (*ITT0* and *ITT1*). For special purposes, data cache is disabled in the address space from 0xFE000000 to 0xFEFFFFFF using the *DTT0* register, but it is enabled and set to copyback mode in the remaining address space (*DTT1*). This ensures maximum CPU performance, since it covers the DRAM address space of this computer system between 0x00000000 and 0x01FFFFFF.

Trouble-shooting

To monitor the effect induced by modifications in the cache setting, the Dhrystone program can be used and its result compared to the numbers given at the beginning of the article. The *cpucache* program can help to analyse the reason for a poor system performance.

- If a 68020, 68030 or 68040 system runs slower than expected, the *cpucache* program should be run without arguments. If it shows all caches to be enabled at least when the system is not in I/O mode, the cache setting is all right but the expectation is probably too high.
- If at least one cache is disabled when the system is not in I/O mode and the disable depth is 0, the appropriate *syscache* module was probably forgotten in the bootlist. The system should behave normally when it is rebooted after generating a new *OS9Boot* file that contains the correct *syscache* module.
- If data cache is disabled during I/O because the *CP2_DDIO* compatibility bit is not set, this normally does not explain a poor performance, since the overall performance will only increase by about 10 to 15% when this compatibility bit is set. On the other hand, the board manufacturer normally has good reasons to deliver a configuration module that has this bit not set.
- If at least one cache continuously shows a disable depth of more than 0 or even shows an increasing disable depth when the *cpucache* program is started repeatedly, there are programs running on the system that make inappropriate calls to the *F$CCtl* function. It is recommended to configure a minimal system and to check the cache setting every time an additional program is started so that the offensive program can be identified.
- If on a 68040 system copyback write cache is not enabled, either no or an inappropriate *ssm* module is used. The system should behave normally when it is rebooted after generating a new *OS9Boot* file that contains the correct *ssm* module.

Hints for Modifying the Cache Control Register and Using the *F$CCtl* Function

There is an undocumented restriction for both modifying the cache control register directly and using the *F$CCtl* function: any operation that affects the setting of a particular cache mode cannot be done unless the cache is flushed prior to modify it. Otherwise, the processor "hangs" or the program exits with strange error messages such as address trap error or similar. The following code segment is, for example, needed to disable data cache in a driver:

```
  vsect
Cache ds.l 1
  ends

  psect cachefunc,0,0,0,0,0

disable_dc:
  movec cacr,d0                    ; get cache control register
  move.l d0,Cache(a2)              ; save it
  andi.l #$7fffffff,d0             ; unset data cache enable bit
  nop                              ; only required for old 68040 mask
```

```
    cpusha dc
    nop                                 ; give the processor time to write
    movec d0,cacr                       ; disable data cache
    nop
    rts
```

Data cache is reset to the initial mode using the following code:

```
reset_dc:
    move.l Cache(a2),d0                 ; get initial cache register setting
    nop                                 ; only required for old 68040 mask
    cpusha dc
    nop
    movec d0,cacr                       ; restore initial cache setting
    nop
    rts
```

The above restriction is also important for using the *F$CCtl* function: if any value other than 0 (flush both caches) is passed to this function, bit 2 and bit 6 must always be set for data and instruction cache actions, respectively, even if the application does not require the particular cache to be flushed.

Compatibility

There is one OS-9 upgrade change from V2.4 to V3.0 being important in the context of the *cpucache* program: The *compat2* bit *CP2_DDIO* is no longer supported. In consequence, there can be no difference in the cache setting between normal mode and during I/O. If, however, this restriction is considered, the *cpucache* program can be used safely under OS-9 V3.0 as well.

References

1. MC68030 Enhanced 32-Bit microprocessor user's manual, 3rd edition, Prentice-Hall, New Jersey

2. MC68040 MC68EC040 MC68LC040 Microprocessors User's Manual, Motorola Inc.

The complete source code environment including also other required files not shown here such as the Dhrystone benchmark program is available on the OS-9 International code disk.

Carsten Emde can be reached by email at <carsten@effo.ch>.

_getsys();

Reto Peter

Monthly EFFO Meetings

The meeting always takes place every first Friday of a month. We meet in Brugg, Hotel Rotes Haus, either in the restaurant (at 7 PM) or in our meeting room in the basement (at 8 PM).

Everybody interested in OS-9 is kindly invited to join the meeting.

OS-9 FAQ

FAQ or Frequently Asked Questions is a common approach to distribute information especially helpful for novice users. The OS-9 FAQ has recently been revised by Boisy Pitre with the help of many OS-9 users. The last but one version is available via anonymous ftp from

chestnut.cs.wisc.edu

in the pub/TEXTS directory under the file name os9faq13. The latest version can be inspected through World Wide Web at

http://www.cs.wisc.edu/~pruyne/os9faq.html

OS-9 International PD-Disk

One of the goals of OS-9 International is to help programming in OS-9. Therefore, many articles contain source code of test and example programs and procedures. In order to facilitate the use and the study of these examples, OS-9 International has decided to make them available in electronically form. The medium chosen is the same as for other software available from EFFO, the PD disk. Hence, from now on, it is possible to order the OS-9 International PD #121.

This disk is updated every time a new issue of OS-9 International appears. This means that the disk will be supplemented with the software from the new issue but all material from previous issues is still available.

For ordering this PD disk, please refer to the EFFO PD list.

Books and Literature about OS-9

The following books about OS-9 are available:

Peter Dibble
OS-9 Insights
Second Edition 1992
Microware Systems Corp.
1900 NW 114th Street
Des Moines, Iowa 50325-7077
USA
ISBN 0-918035-03-1

Paul S. Dayan
The OS-9 Guru, 1-The Facts
Galactic Publication 1992
Galactic Ind. Ltd.
Unit 3B
Mountjoy Research Centre
Stockton Road,
Durham, DH1 3UR
United Kingdom
ISBN 0-9519228-0-7

Microware
The OS-9 Catalog,
1992/1993 edition
Microware Systems Corp.
1900 NW 114th Street
Des Moines, Iowa 50325-7077
USA

The third edition of Peter Dibble's book is already in press. It should be available in Europe before end of this year. In addition, an article entitled *David versus Goliath* was published in the September '94 issue of the Wired magazine. The article contains a lot of background information about the history of OS-9 and the Digital Audio/Video Interactive Decoder (DAVID). It can be inspected through World Wide Web at

http://www.hotwired.com/Lib/Wired/2.09/departments/electrosphere/microware.html

Announcement of the EFFO AGM 1995

The EFFO Annual General Meeting 1995 will be held at

Saturday, 21. January 1995 in the restaurant Herberge in Teufenthal, Switzerland.

The meeting will start at 2:30 PM. A detailed invitation will be sent out to EFFO members before end of the year.

Imprint

OS-9 International

Published by	European Forum For OS-9 (**EFFO**)
President	Werner Stehling
Vice President	Reto Peter
Director of Finance	Stephan Paschedag
Editor-in-Chief	Carsten Emde
Design	Marc Balmer
Layout	Werner Stehling

Address
European Forum For OS-9
P.O. Box
8606 Greifensee
Switzerland

FAX +41 1 940 38 90
email os9int@effo.ch

Printed in Switzerland
ISSN: 1019-6714

Subscriptions

OS-9 International is the official organ of the European Forum For OS-9 (**EFFO**). The subscription is included with the annual **EFFO** membership fee. In addition, it is available by separate subscription for non-**EFFO** members, single issues are also available. All following prices are given in Swiss Francs, shipping included:

	Switzerland	*Europe*	*Overseas*
One year (3 issues)	25.00	30.00	35.00
Single issue	10.00	12.00	14.00

To subscribe to **OS-9 International** or to order a single issue send a letter, postcard, fax or email to **EFFO**.

Advertisements

OS-9 International is not only an ideal platform for discussing OS-9 related topics, it is also the ideal place to advertise. **OS-9 International** reaches end-users, system-software developers and, nevertheless, decision-makers.
Please contact **EFFO** for detailed information on how to place an ad in **OS-9 International**.

EFFO *European Forum For OS-9*
8606 Greifensee
Switzerland
Fax +41 1 940 38 90

What is EFFO and who should be interested in it?

EFFO stands for *European Forum For OS-9* and was founded in 1988; its main goal is to support Microware's multi-user multi-tasking operating system OS-9 that runs on the 68k family of Motorola microprocessors. This support primarily consists in providing a means of communication between people who already use and appreciate OS-9 and those who do not yet but would profit by doing so.

EFFO is independent from and not commercially related to any company. Its members are companies offering OS-9 compatible hardware, OS-9 system programmers, computer clubs as well as end users such as private computer owners, research institutes and university departments.

EFFO intends

- to provide a collection of public domain software that is of general interest and that helps to make OS-9 more attractive to programmers and users,
- to coordinate ports of (mainly UNIX) software (e.g. to avoid that a particular software is ported many times and other equally important software still is not),
- to make all the nuts and bolts of managing an operating system available to everybody so that "the wheel has not to be re-invented all the time".

Until the end of 1992, the rules of the game were that disks containing the EFFO software pool were available without charge, and everybody taking part in this service was expected to make contributions to the pool. Initially, this concept – although somewhat idealistic – worked quite well. Over the years, however, less and less contributions have been made so that, starting in 1993, a new concept was created.

First the bad news: The distribution is no longer free of charge, there is a handling fee in an order of magnitude comparable to all other PD pools still being incompatible to OS-9.

And now the good news:

- The disks contain ready-to-use software that has been thoroughly tested.
- The software is maintained and updated continuously.
- All disks come with printed guidelines of how to install and to use the software. Some of them even have a complete user's manual in printed form.
- The 23 forum editions and the 10 PD disks representing the EFFO software pool as at end 1992 will – although no longer maintained and upgraded – still be available.

In addition, EFFO has a printed forum: the journal *OS-9 International*, published by EFFO, is devoted to OS-9 related topics. Every issue includes the most recent version of EFFOs public domain software list. This list will also be made available on the EFFO bulletin board and through regular postings to international network boards.

Please contact EFFO at

EFFO
P.O. Box
CH-8606 Greifensee
Switzerland
Fax +41 1 940 38 90
email: effo@effo.ch

to obtain more information. This is also the address where the editorial staff of *OS-9 International* and all active EFFO members can be reached in case of questions concerning particular articles or software packages.

Last but not least we invite you to join EFFO. As a regular member you get some price reduction on our PD disks, and a one-year subscription to *OS-9 International* is included. Your membership will be very helpful not only in financial aspects but also as moral support: a bigger EFFO can move bigger mountains, can't it? For enlistment of three new EFFO members you get a free membership for the next year.

CR, WS 94/12

Notes concerning the PD collection

- The updated list is regularly published in *OS-9 International*.
- Version numbers normally reflect the number assigned by the author. Collections of several distinct items get a version number assigned by EFFO. Usually we deliver the newest versions available.
- The disks are 3.5" in universal format. Normally OS-9 version 2.4 or higher will be required.
- Normally the program is free for personal use. In general, the GNU licence or the copyright notice of the author must be respected.
- Prices are calculated on a per disk base. The units are Swiss Francs or Deutschmark (to keep things simple we use a 1:1 exchange rate).

EFFO OS9 **European Forum For OS-9**
8606 Greifensee
Switzerland
Fax +41 1 940 38 90

EFFO PD Collection — Order Form

valid as from **2 Dec 94**

Nr.	Title	Vers.	Contents	Status	# of Disks	Order
100	Utilities	1.1	*diff* compare files; *upatch* inverse to diff; *fgrep; mkdir* builds complete path; *move; space, dinfo, rendisk: disk utilities; help:* like VAX/VMS; *cheksum, fillup, rawcopy, exb:* EPROM utilities		1	
101	SC	6.21	*sc* spreadsheet calculator		1	
102	GCC	1.42.0	GNU C compiler (in compliance with ANSI), executables only; about 2 MB RAM needed		1	
103	GCC sources	1.42.0	GNU C compiler, sources		3	
104	GPP	1.40.3	GNU C++ compiler, executables only		2	
106	Ghostscript	3.12	Postscript level 2 interpreter for output devices such as various HP printers, other printers, MGR window manager, GIF/PCX/PBM/PGM/ PPM file formats, etc.	update to level 2	4	
107	Shell	1.7	*sh* combined and enhanced features of the Microware and the Bourne shell, full VT100 support	update	1	
108	Communication	1.1	C–*Kermit* 5A(188); *inout* connects two SCF devices; *Z–Modem* 3.17		1	
109	Network	1.0	*SLIP* (ex KA9Q): TCP/IP via RS232		1	
112	OS–9 Lib	90/3/18	set of procedures available on UNIX-systems to allow and ease implementation of UNIX-software on OS–9 systems		1	
113	Communication source	1.1	*C–Kermit* source 5A(188), *Z–Modem* source 3.17, executables are on PD#108		2	
117	GCC	2.5.8	GNU C compiler (in compliance with ANSI), executables only; about 4 MB RAM needed		2	
119	GPP LIBGPP	2.5.8 2.5.3	GNU G++ compiler, LIBG++, executables only; about 4 MB RAM needed		4	
121	OS-9 International	1.0	programs that appeared in OS-9 International and accompanying programs; covers up to and including issue 3/94	new	1	
123	OS9exec	1.0	OS-9 cross development on the Macintosh, single task, no realtime. **Note:** this is a Macintosh disk	new	1	
					total number of disks	

				Unit Price	Qty	Price
PDs	Software		total number of PD disks (normal / EFFO member price)	12.— / 8.—		
DOC	Docu box	1.0	hardcover folder with box to collect manuals (A5 size)	30.—		
HDL	handling	1.0	package and handling (add with disk or box orders only)	10.—	1	
ESM	single member	1994	EFFO single membership (private users)	80.—		
EGM	group member	1994	EFFO group membership (companies, institutions, computer clubs, users groups)	150.—		
SUB	subscription	1 year	separate 1-year subscription to *OS–9 International* (3 issues). The three prices are for shipping to Switzerland, Europe or Overseas, respectively.	25.— 30.— 35.—		
					Total Amount (SFr / DM)	

Name .....

Address .....

City .....

Country .....

Phone .....

Fax .....

Date / Sign .....

Sorry we don't accept credit cards or cash. Add 5% for checks. Please prepay to one of the following accounts:

Switzerland:
Postal Giro Account, 80–48 254–4 (Zürich)

Germany:
Volksbank Jestetten eG, 111 2007 (BLZ 684 915 00)

❑ Please bill me in advance

❑ I am already EFFO member

my name and address is

❑ confidential ❑ for members only ❑ free